VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Aplikace pro správu webhostingu a multihostingu pro platformu Android

Management Application for Webhosting and Multihosting for Android Platform

2014 David Furmánek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **David Furmánek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Aplikace pro správu webhostingu a multihostingu pro platformu Android
Management Application for Webhosting and Multihosting for Android Platform

Zásady pro vypracování:

Cílem práce je navrhnout a implementovat aplikaci, která umožní provádět správu webhostingových a multihostingových služeb provozovaných pod značkou SvetHostingu.cz na platformě Android. Aplikace musí vzdáleně komunikovat se serverem provozovatele, přes který budou přijímány a odesílány požadavky na správu služeb. Popis rozsahu spravovaných služeb je uveden v bodu 4 zadání.

1. Seznamte se s aktuální problematikou v oblasti vývoje aplikací pro platformu Android.
2. Seznamte se s webhostingovými a multihostingovými službami provozovanými pod značkou SvetHostingu.cz.
3. Navrhněte a otestujte způsob komunikace mezi vyvíjenou aplikací a serverem provozovatele.
4. Navrhněte strukturu a jednotlivé části vyvíjené aplikace. Aplikace musí umožňovat správu webhostingových a multihostingových služeb v následujícím rozsahu: přihlášení, správu e-mail účtů, správu fakturačních a kontaktních údajů, přehled uhrazených a neuhrazených faktur, monitoring zátěže (cpu, ram, mysql, procesů a místa na disku). Pomocí aplikace dále musí být možné provést objednání nových služeb. Mezi tyto služby patří: objednání webhostingu nebo multihostingu a registrace domén.
5. Proveďte implementaci navržené aplikace na platformě Android.
6. Otestujte nejprve na testovacích a následně reálných datech funkčnost a bezpečnost celé aplikace.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Libor Holub, Ph.D.**

Konzultant bakalářské práce: Mgr. Ing. Michal Krumnikl

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

Zdrojové kódy aplikace a návrh struktur protokolu jsou výlučným vlastnictvím zadavatele, a mohou být zpřístupněny pouze s písemným souhlasem zadavatele. Zadavatel uděluje právo přístupu akademickým pracovníkům a členům zkušební komise, kteří budou provádět hodnocení studenta.

Libor Holub
NETtip
Krakovská 3
700 30 Ostrava 3
tel./fax: 596 716 847
info@katalogtip.cz
IČO 70311919 DIČ CZ7901045581

V Ostravě 30. března 2014 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. března 2014 .....

Rád bych poděkoval panu Ing. Liboru Holubovi, Ph.D. za udělenou možnost, cenné rady a důvěru do mne vloženou při vypracování této práce. V neposlední řadě také mým rodičům kteří mne vždy podporovali.

Abstrakt

Tato práce popisuje návrh a implementaci aplikace pro operační systém Android podle zadání firmy SvetHostingu.cz. V práci jsou analyzovány často používané mobilní operační systémy a zdůvodnění výběru platformy Android. Zabývá se také možným použitím nějakého existujícího frameworku, který se zaměřuje na administraci serveru, a proč nebyl použit. Softwarový návrh se zabývá architekturou aplikace a použitím model-view-controller návrhového vzoru pro komunikaci se serverem a je zde navržena struktura použitého komunikačního protokolu. Implementace popisuje nejdůležitější části aplikace a problémy, které vyvstaly při programování a jejich řešení. Aplikace je v současnosti dostupná na Google Play.

Klíčová slova: Android, Java, SOAP, MVC, Hosting, Webhosting, Multihosting, Server, síťová komunikace

Abstract

This thesis describes the design and implementation of application for Android operating system according to the instructions of the SvetHostingu.cz company. In this thesis there are analyzed frequently used mobile operating systems and the reason why Android was selected as a platform. It also concerns with the potential use of an existing framework, which focuses on server administration and why it was not used. Software design deals with architecture of application and use of a model-view-controller design pattern for communication with the server and there is suggested a structure of the communication protocol. The implementation describes the most important parts of the application and problems that emerged during programming and their solutions. The application is currently available on Google Play.

Keywords: Android, Java, SOAP, MVC, Hosting, Webhosting, Multihosting, Server, network communication

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
APK	–	Android application package file
CPU	–	Central Processing Unit
HTTP	–	Hypertext Transfer Protocol
HTTPS	–	Hypertext Transfer Protocol Secure
JSON	–	JavaScript Object Notation
JVM	–	Java Virtual Machine
OS	–	Operační systém
PDF	–	Portable Document Format
RAM	–	Random-Access Memory
SDK	–	Software Development Kit
SOAP	–	Simple Object Access Protocol
SQL	–	Structured Query Language
TLD	–	Top Level Domain
UI	–	User Interface
XML	–	Extensible Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V dnešním světě se stalo používáni tzv. chytrých telefonu naprosto běžnou věcí. Lidé používají svůj mobilní telefon jako kapesní počítač pro zábavu i práci a zvládnout na něm stejné úkony jako na stolním počítači. Z tohoto důvodu se zadavatel, tedy firma SvetHostingu.cz, rozhodl vytvořit aplikaci pro mobilní zařízení, která by zjednodušila práci s jeho službou.

Firma SvetHostingu.cz se zaměřuje na poskytování hostingových služeb, tedy hostování osobních i firemních webových stránek. Existují dva hlavní programy poskytovaných služeb a to *webhosting* a *multihosting*. Základní rozdíl z pohledu uživatele je, že zatímco webhosting má určitý maximální počet domén, multihosting má neomezený počet domén a podle zvoleného platebního programu může uživatel určit hardwarové parametry. Multihostingové řešení je totiž druh virtuálního stroje, uživatel si tedy kupuje určitý výkon a kolik webů zde poběží už záleží na uživateli samotném. Jak rozdílně aplikace musí přistupovat k těmto dvěma druhům účtům je popsáno v kapitole 2.

Aplikace samotná komunikuje se serverem služeb SvetHostingu.cz a v kapitole 3 je popsáno jakým způsobem tato komunikace probíhá. Uvedené jsou zde běžné způsoby přenosu dat a analýza existujících systému, které řeší podobný problém.

Porovnání existujících mobilních operačních systémů a důvod proč byla pro tuto aplikaci vybrána platforma Android je v kapitole 4.

V kapitole 5 je popsán softwarový návrh aplikace, tedy funkční specifikace, architektura a struktura komunikačního protokolu.

Předposlední kapitola 6 je popis implementace podle softwarového návrhu. Kapitola se věnuje řešením síťového přenosu, manipulaci s daty a implementaci uživatelského rozhraní.

Práci zakončuji popisem průběhu testování aplikace v kapitole 7.

2 Specifikace zadání

2.1 Požadavky na aplikaci

Hlavním smyslem aplikace je rychlá a přívětivá správa účtů v systému svethostingu.cz pomocí mobilního zařízení.

Uživatel webhostingu má možnost:

- Provést autorizovaný požadavek, který se odešle podpoře.
- Spravovat poštovní účty – vytváření nových účtů a zobrazit jejich přehled, přidávat přeposílání a přesměrování na vytvořené účty a nastavit doménový koš.
- Fakturace – detailní přehled uhrazených/neuhrazených faktur s možností stažení faktury v PDF.

Uživatel multihostingu má navíc tyto možnosti:

- Aktivace služeb – aktivace dalšího webhostingu v rámci multihostingu, to znamená registrace domény nebo alias domény s možností přesměrování na již existující webhosting(doménu). Možnost převodu domény od jiného registrátora.
- Monitoring zátěže – zobrazení grafů zátěže nebo využití v čase. Lze zobrazit zatížení CPU, využití pamětí RAM, dotazy na MySQL, počet připojení k MySQL, počet procesů, obsazené místo a využití i-nodů. Statistiky je možné zobrazit denní, týdenní, měsíční a roční.
- Ověřit dostupnost domény a v případě, že je volná nabídnout možnost registrace.

Aplikace dále musí umožňovat registrovat nový multihostingový účet. Registrace domény pak probíhá v *Aktivaci služeb* po přihlášení. Aplikace podporuje i mojeid.cz pro přihlášení a registraci v systému svethostingu.cz.

3 Způsob komunikace mezi serverem a aplikací

Komunikace je typu klient-server a probíhá mezi aplikací(klient) a serverem. Klient pošle dotaz na server, který vyhodnotí požadavek v dotazu klienta a vrátí odpověď, kde se klient dozví zda se požadavek provedl v pořádku.

V sekci 3.2 jsou popsány některé existující způsoby přenosu dat.

3.1 Server

Server sloužící jako koncový bod, na nějž aplikace posílá své dotazy je naprogramován v jazyce PHP. Komunikace je vedena textově prostřednictvím bezpečného protokol HTTPS, do kterého jsou vloženy data obsahující požadavek.

3.2 Možné způsoby přenosu dat

HTTP, respektive HTTPS, je textový protokol, a tak i výběr možných způsobů přenosu dat se zaměřuje na možnosti HTTP. Přenos binárních dat je opomenut z důvodu jeho nevhodnosti a zbytečné složitosti pro potřeby aplikace. Pro přenos dat byly zvažovány protokoly *JSON*, *XML* a *SOAP*.

3.2.1 JSON

JSON(JavaScript Object Notation) je jednoduchý způsob serializace dat a tím je umožněn jejich snadný přenos mezi klientem a serverem. Velkou výhodou pro mobilní aplikace je jeho relativně malá velikost. Počítač jej umí dobře zapisovat i číst a je čitelný i pro člověka. I přes jeho název jej lze použít napříč různými programovacími jazyky. Nevýhodou může být nemožnost nastavení kódování znakové sady. [1]

```
[true,1,"string",{"a":"a"}]
```

Výpis 1: Ukázka serializace JSON

3.2.2 XML

Extensible Markup Language je značkovací jazyk, zapisovaný ve stromové podobě, který se používá pro popis dat. Je zapisován v textové formě, čitelné i pro člověka. Názvy tagů (značek) v něm lze vytvářet dle libosti a je možné takto popsat jakýkoliv datový formát nebo objekt. XML je hojně využívaný pro přenos dat mezi různými systémy a programovacími jazyky. Nevýhodou XML je velká "ukecanost", která zvětšuje velikost

přenášených dat a teoreticky libovolně velké zanoření stromu může způsobovat problémy na pomalejších zařízeních. [2]

```
<?xml version="1.0" encoding="UTF-8" ?>
<user timeout="1800">
    <username>login</username>
    <password>password</password>
</user>
```

Výpis 2: Ukázka XML formátu

3.2.3 SOAP

SOAP(Simple Object Access Protocol) je na XML založený protokol primárně určený pro výměnu zpráv přes HTTP. SOAP zjednodušuje komunikaci mezi klientem a serverem tak, aby nezáleželo na platformě ani jazyku (zaručuje XML) a zároveň umí popsat přenášené data jakéhokoliv jazyku. SOAP navíc umožňuje zpracování neočekávaných chyb. Klientovi vždy přijde zpět SOAP odpověď, byť se sdělením chyby místo dat. SOAP je tvořen envelope(obálkou). V obálce je header(hlavička) a body(tělo – data). Je tedy ještě objemnější pro přenos po síti, než prosté XML. [3]

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <login xmlns="http://soap.svethostingu.cz/">
            <username>login</username>
            <password>password</password>
        </login>
    </soap:Body>
</soap:Envelope>
```

Výpis 3: Ukázka SOAP formátu

3.3 Zvolený protokol

Z pohledu vývoje pro Android se jako nejlepší jeví protokol JSON, pro jeho malou velikost a jednoduchou implementaci, protože je již nativně v Android SDK podporován a existují pro práci s ním Java knihovny. Podobně je na tom i formát XML, který je také nativně Androidem podporován.

Po všeobecném zhodnocení možností i z pohledu serveru, byl nakonec vybrán protokol SOAP. Důvodem je, že v systémech svethostingu.cz se již používá pro komunikaci mezi různými částmi systému a server, na který bude aplikace posílat své dotazy, je na tento protokol připraven.

3.4 Průzkum existujících systémů

Na místě je otázka zda neexistuje nějaké univerzální řešení, které bychom mohli použít třeba i na více platformách a nebylo by potřeba programovat aplikaci od nuly. Existuje i odvětví s názvem *mobile enterprise application platform*, které se zaměřuje na zjednodušení vývoje aplikací pro firmy. Tyto řešení jsou často multiplatformní, fungují tedy na široké škále mobilních zařízení, ovšem často za cenu výkonu aplikace. Používají generování kódu, který může být neefektivní, nebo využívají HTML5+JavaScript či podobné „triky" jak jednoduše vyvíjet multiplatformně.

Za zmínku stojí:

- SAP Mobile Platform[1] – dříve známá jako *Sybase Unwired Platform* je řešení, které vytvoří mezivrstvu(middleware) pro zpracování dat ze serveru, zaměřuje se hlavně na relační databáze, enterprise aplikace, soubory a jejich čtení a zápis na mobilním zařízení a odeslání zpátky na server. Tato platforma není vhodná zejména z důvodu větších nároků na server, tedy použití nějakého řešení firmy SAP, nebo dodržení jejich standardů pro odesílání dat. Problémem je také cena, kdy se prodávají balíčky pro použití určitého počtu zařízení, to je vhodné pro interní firemní použití, ne pro klienty služeb svethostingu.cz. Další možností je procentuální platba ze zisku v použité aplikaci. Je otázka zda je, nebo není tato aplikace zdarma, protože užitek z ní mají jen platící klienti služeb svethostingu.cz.

- OpenMEAP[2] – tato platforma používá HTML5+JavaScript a je tedy stavěná na tom, že dnešní mobilní zařízení umí zobrazit webové stránky a ty používá pro vykreslení UI. Tyto aplikace obecně trpí horším „pocitem uživatele". UI má často špatnou odezvu na akce uživatele a působí „zpomaleně". OpenMEAP je opensource projekt poskytovaný zdarma, který podporuje Android, iOS a BlackBerry.

Obecně lze říci, že žádné z těchto řešení není vhodné z důvodu požadavku na co nejmenší závislost aplikace na třetích stranách. Pokud bychom použili nějaký projekt a jeho vývoj se časem zastavil, bylo by nutné aplikaci přepsat nebo pokračovat v používání frameworku bez možnosti reagovat na nové funkce operačních systémů.

[1] `http://www.sapmobile-platform.com/`
[2] `http://www.openmeap.com/`

4 OS Android

Operační systém Android je v současnosti nejrozšířenější mobilní platforma[3]. Je to open source platforma založená na Linuxu (jádro systému), kterou může použít na své vlastní zařízení jakýkoliv výrobce. Android vyvíjí Open Handset Alliance, což je sdružení společností, které se snaží o vývoj otevřených standardů v oblasti mobilních zařízení. Členy jsou firmy jako Google, HTC, Sony, Intel, Samsung, T-Mobile, Nvidia a další. Vedoucí firmou a zároveň iniciátorem sdružení je Google.

4.1 Proč byl vybrán Android

V době začátku vývoje aplikace (začátek roku 2012) bylo uvažováno o dvou platformách, pro jejich největší rozšířenost a předpokládané udržení na trhu. Byly to operační systémy *Android* od firmy Google a *iOS* od firmy Apple. V příloze A je graf vývoje podílu mobilních operačních systémů na trhu v intervalu od února roku 2011 do února roku 2012. Je zde vidět skoro až padesáti procentní zastoupení Androidu a kolem dvaceti procent systému iOS.

Další kritérium byl levný vývoj a rychlé seznámení s platformou. Google poskytuje Android SDK a své vlastní vývojové prostředí zdarma, což je i případ firmy Apple. Avšak pro vývoj na iOS je potřeba mít počítač s operačním systémem Mac OS X[4], jelikož iOS SDK a jeho vývojové prostředí běží pouze pod tímto systémem. Naproti tomu aplikace pro Android lze vyvíjet v podstatě na jakékoliv platformě[5], protože SDK i vývojové prostředí pracují v JVM a jsou tak spustitelná všude kde funguje Java. Dalším možným problémem u iOS je registrace vývojáře, která činí $99 na rok. Tento poplatek zpřístupní testování na reálném zařízení s iOS a možnost její publikace na App Store. U Androidu je možno vyvíjet a testovat aplikace na reálném zařízení prakticky bez omezení. Registrace vývojáře a s ní spojený jednorázový poplatek $25, je potřeba jen pro publikování aplikace na Google Play. Pro Android se vyvíjí v jazyku Java, pro iOS se používá jazyk Objective C.

Oba systémy se dají porovnat i na základě různorodosti zařízení ve kterých se používají, především však rozlišení displeje. Zatímco zařízení se systémem Android mají různé velikosti displejů a k nim různé rozlišení. Zařízení s iOS je jen pár druhů a velikosti displejů tolik neliší. To umožňuje mnohem snadnější testování vzhledu aplikace, protože si můžeme být jistí rozlišením a celkově aplikaci designovat pro konkrétní rozlišení. Velká variabilita Android zařízení činí testování obtížnější, jelikož je třeba zkoušet různé velikosti displejů, jejich rozlišení a s tím spojené DPI, podle kterého systém určí použitou sadu

[3] http://en.wikipedia.org/wiki/Android_(operating_system)#Market_share
[4] https://developer.apple.com/programs/ios/
[5] https://developer.android.com/sdk/

obrázků na konkrétním zařízení. Je tedy potřeba vytvořit používané obrázky v různých velikostech.

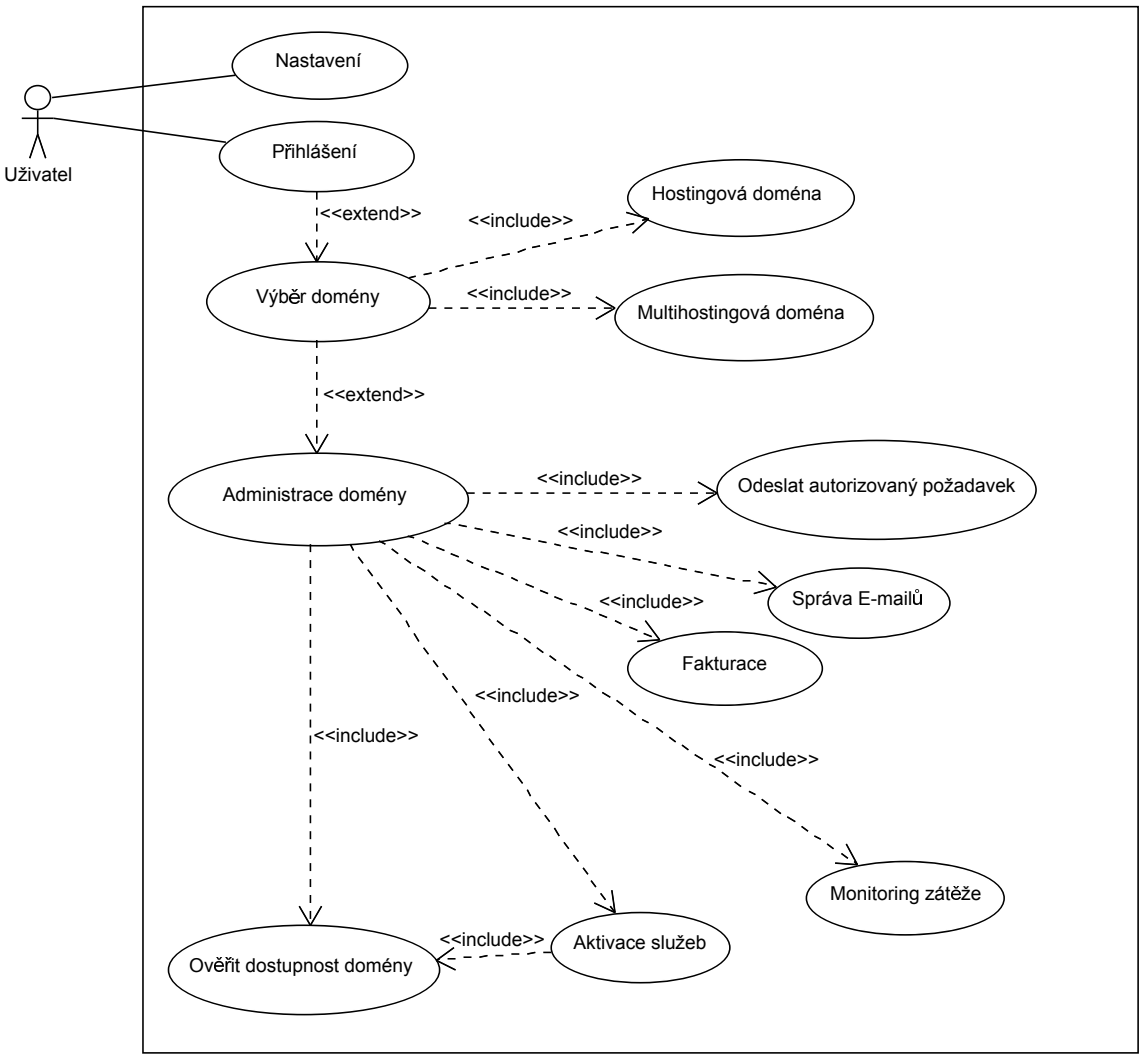
Z výše uvedených výhod a nevýhod každé platformy jsme, po dohodě se zadavatelem, vybrali systém Android, zejména pro možnost bezplatného testování aplikace na reálném zařízení, menší investici do hardwaru na vývoj a také z důvodu větší dostupnosti mobilních telefonů s Androidem, který vlastní zadavatel. Důvodem je také větší rozšířenost na trhu a předpoklad že klienti služeb svethostingu.cz budou mít k dispozici Android zařízení. Také moje znalost jazyka Java byla výhodou a osobně je mi přístup společnosti Google přívětivější. Více informací o obou systémech na webech [6, 7].

Operační systém iOS je nezanedbatelnou částí trhu. V případě úspěchu aplikace a zájmu mezi klienty služeb svethostingu.cz bude implementace pro iOS zvažována.

5 Softwarový návrh aplikace

5.1 Funkční specifikace

V kapitole 2 je specifikováno zadání co vše aplikace musí splňovat. Nezabývá se však přesnými detaily fungování. Uživatel se například musí nejprve v aplikaci přihlásit než může začít spravovat svůj účet. Případ užití pro přihlášení a následné pokračování k vlastní správě účtu ukazuje obrázek 1.



Obrázek 1: Hlavní případ užití

Po přihlášení si uživatel musí zvolit doménu, kterou bude spravovat. V systému svethostingu.cz je možno mít ve svém účtu dva druhy hostingu, Webhosting a Multihosting. Webhosting je obyčejný hosting s přiděleným webovým prostorem a určitým počtem domén. Multihosting na druhou stranu umožňuje mít neomezeně domén, protože uživatel si pronajímá webový prostor, který je virtualizován. Má kontrolu nad nastavením výkonu CPU, velikosti RAM a podobně. Z tohoto důvodu má multihostingový účet více možností administrace. Níže se budu zabývat požadavky na části aplikace, které umožňují uživateli správu účtu.

5.1.1 Přihlášení

Samozřejmostí je samotné přihlášení, tzn. je po uživateli vyžadováno uživatelské jméno a heslo. Pro zpříjemnění přihlašování, by měl mít uživatel možnost přihlašovací údaje uložit pro příští použití. Zde je nutno podotknout, že pro uložení těchto údajů je třeba analyzovat možnosti bezpečného ukládání dat v Androidu. Ukládání přihlašovacích údajů umožní snadnější přihlášení a práci mezi více uživatelskými účty.

Při použití služby *mojeid.cz* je třeba zajistit komunikaci s touto službou. Mojeid.cz poskytuje návod [6] jak mojeid.cz implemenotvat. Systém svethostingu.cz službu implementuje, ale pro tuto aplikaci je třeba provést úpravy.

5.1.2 Výběr domény

Výběr domény, jak už bylo zmíněno výše, musí být rozdělen na *Webhosting* a *Multihosting*. Uživatel může mít pod jedním účtem více multihostingových účtů, proto je navíc třeba ještě zvolit konkrétní multihosting pod kterým jsou až domény určené pro správu.

5.1.3 Administrace domény

V této části se uživatel dostane k vlastní správě domény. Zde je výběr všech možností administrace a pro uživatelovo pohodlí umožnit rychlé přepínání domény. V případě multihostingu je možná volba jen domén pod tímto multihostingovým účtem. *Autorizovaný požadavek*, *správa e-mailů* a *fakturace* jsou zobrazeny jak webhostingu, tak multihostingu. Ostatní jen pro multihostingový účet.

[6] `http://www.mojeid.cz/page/1862/jak-zavest-mojeid/`

5.1.4 Odeslat autorizovaný požadavek

Tato část umožní uživateli napsat zprávu, která se odešle na podporu systému svethostingu.cz. U zprávy umožnit nastavení na jakou e-mailovou adresu má přijít odpověď. Implicitně je vhodné nastavit adresu účtu.

5.1.5 Správa e-mailů

Kromě zobrazení poštovních účtů, jejich přesměrování a přeposílání, je třeba účty vytvářet. Při vytvoření je vyžadován název e-mailu, heslo a velikost schránky, která se pohybuje od deseti megabajtů do dvou gigabajtů. U přidání přeposílání a přesměrování musí uživatel vybrat z existujících e-mailů ve zvolené doméně a zadat na který přesměrovat, či přeposílat. Doménový koš je možné aktivovat nebo deaktivovat, při aktivaci se musí zvolit existující e-mailový účet v rámci domény.

5.1.6 Fakturace

V zadání je uvedeno *detailní přehled uhrazených/neuhrazených faktur*, to znamená zobrazit číslo faktury, datum splatnosti, stav zaplacení a částku. V detailu konkrétní faktury pak navíc datum vystavení a podrobný popis účtovaných služeb. Pro uživatelův komfort je vhodné zobrazit údaje k platbě s variabilním a konstantním symbolem a přehled účtů, na které je možno platbu zaslat a fakturační adresu plátce. Pokud už byla zaplacena tak umožnit stažení pdf souboru s fakturou.

5.1.7 Aktivace služeb

Aktivací služeb se rozumí objednání další domény pod přihlášený účet a je rozdělena na více kroků. Bude obsahovat košík pro možnost odeslání více objednávek najednou.

1. Volba typu aktivace – *webhosting(v rámci multihostingu)*, *alias* s a bez *přesměrování*
2. Vložení domény – uživatel zadá požadovanou doménu, která se před dalším krokem ověří, zda je *dostupná*, nebo je možný *transfer* domény do systému svethostingu.cz.
3. Zde probíhá větvení
    - Doména je volná k registraci – je nutno zadat kontakt, který se k doméně přiřadí.
    - Doména obsazená a je možný transfer – uživatel musí zvolit ze dvou možností. Doména se převede do systému svethostingu.cz nebo se provede pouze aktivace služeb (doména zůstává u stávajícího registrátora).

4. Fakturační kontakt – při aktivaci služeb je třeba zadat fakturační kontakty na které má být služba vyfakturována.

5. Konečný přehled – přehled vybraných možností s cenou. Zde uživatel objednávku potvrdí nebo zruší, případně vloží do košíku.

5.1.8 Monitoring zátěže

Zobrazení zátěže popsané v kapitole 2. Uživatel bude mít možnost dynamicky přepínat grafy a jejich časové intervaly.

5.1.9 Ověření dostupnosti domény

Tato volba byla popsána v části 5.1.7 pod názvem *Vložení domény*. Tato osamostatněná možnost se liší v primárním použití na rychlé ověření dostupnosti domény. Po ověření nabídne uživateli aktivaci.

5.2 Navržená struktura protokolu

Server posílá data v určitém, námi specifikovaném, formátu. Jak už bylo řečeno v kapitole 3, je server naprogramován v jazyce PHP a do SOAPu vkládá data jako asociativní pole, které je nutno v aplikaci zpracovat pro použití v jazyku Java.

Protokol má jednoznačně danou základní strukturu a názvy použitých polí. V indexu jménem *status* je informace, zda požadovaný příkaz byl vykonán bez chyb nebo s chybou. Index jménem *data* pak obsahuje konkrétní informace závisející na tom jaký příkaz jsme na serveru volali. Např. při přihlášení, které proběhlo v pořádku, bude status *ok* a v položce data se bude nacházet *SSID*. Při používání dalších funkci se SSID používá k autentizaci již přihlášeného uživatele. V případě chyby je v položce data obsažena informace o chybě, tedy číslo chyby a textový popis. Data posílaná na server mají pouze položku *data*.

```
[data] => Array
    (
        [login] => username
        [password] => tajneheslo
    )
```

Výpis 4: Data posílaná na server při přihlášení

```
[status] => ok
[data] => Array
    (
        [ssid] => 282fd0871f79a65891d08c9bb988e2b39915cdd8b7e1bf1a89a
    )
```

Výpis 5: Data přijímaná ze serveru při přihlášení

5.3 Architektura aplikace

Všechny výše uvedené části aplikace musí nějakým způsobem komunikovat se serverem. Jako základ námi používaného protokolu byl zvolen *SOAP*, který však není v Androidu standardně přímo podporován. V kapitole 5.4 je popsáno řešení.

Pro implementaci byl zvolen jako základní návrhový vzor **Model-View-Controller**, který odděluje zpracování a odesílání dat od zobrazení a interakce uživatele. Tento přístup zpřehledňuje aplikaci a zejména hlavně umožňuje relativně jednoduchou výměnu jednotlivých částí. Například pokud bychom změnili komunikaci na jiný protokol než SOAP, není třeba vůbec přepisovat žádné UI ani funkce v třídách starající se o zobrazování dat.

5.3.1 Model

Modelem se rozumí data aplikace a práce s nimi. V tomto konkrétním případě se tato vrstva stará i o komunikaci se serverem, zpracování odpovědi serveru a uložení dat do paměti v konkrétním formátu.

5.3.2 View

View je vrstva grafické prezentace. Představuje vzhled aplikace, který uvidí uživatel, rozložení komponent a podobně. Její hlavní úkol je zobrazit data z modelu v UI.

V Androidu je *View* reprezentován XML souborem, ve kterém je zapsáno rozvržení komponent UI. Řeší se zde i podpora pro různé rozlišení a velikost displejů nebo změna rozložení UI v horizontální či vertikální poloze zařízení. Android totiž umožňuje mít těchto souborů, které popisují stejné UI, více a jejich použití podle konkretního zařízení řeší často automaticky.

5.3.3 Controller

Contoller se stará o komunikaci mezi vrstvami *View* a *Model*. Reaguje na požadavky uživatele, které přicházejí z View a pokud je to třeba, zavolá Model aby připravil data

nebo naopak data odeslal. Controller reaguje i na změny v Modelu a podle toho aktualizuje View vrstvu.

V Androidu je typický Controller třída `Activity`, případně `Fragment` jak je popsáno v kapitole 6.

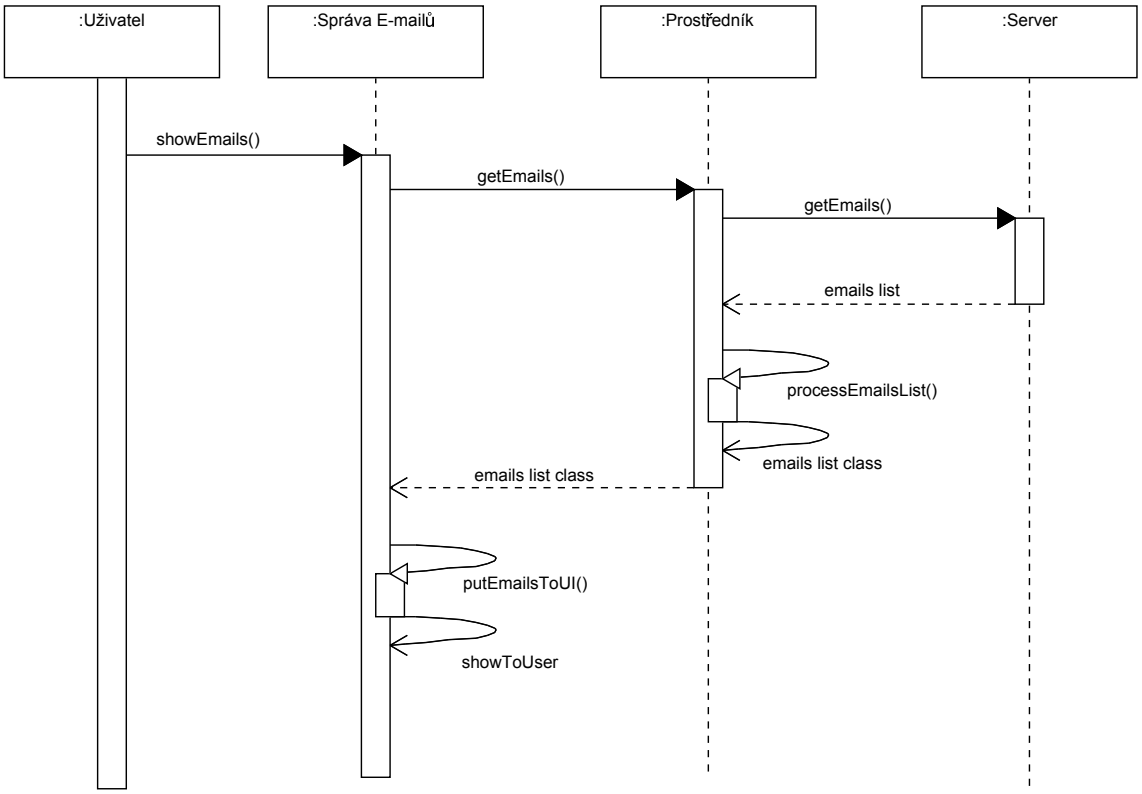
5.3.4 Komunikace se serverem

Klient se serverem si vyměňují jednoduché zprávy, klient buď požaduje nějaká data pro zobrazení uživateli nebo naopak serveru posílá data, která uživatel změnil a vyžaduje jejich zpracování. Komunikaci vždy zahajuje klient a čeká na odpověď serveru. To znamená, že neexistuje potřeba žádných upozornění pro uživatele když aplikace neběží. Uživatel ovládá chování aplikace přímo a vše probíhá jen pokud je aplikace v popředí.

Jak bylo nastíněno v kapitole 5.2, probíhá přihlášení uživatele jednou. Poté aplikace obdrží *SSID*, které slouží jako autentizace uživatele a jeho požadavků. Tedy při volání dalších funkcí je vždy vyžadováno SSID, které identifikuje instanci přihlášení. U některých funkcí, které jsou dostupné i bez přihlášení, např. vytvoření nového účtu, je vyžadován místo SSID, jiný speciální String. Tento řetězec slouží pouze k tomu, aby nešlo jednoduše posílat nesmyslné dotazy na server zvenčí. Pokud při vytváření nového účtu server nedostane správný řetězec, odmítne účet vytvořit.

V rámci návrhového vzoru *MVC* je potřeba tuto komunikaci oddělit od zbytku aplikace. Protože je Java objektový jazyk, komunikace se serverem bude probíhat pomocí jednoho objektu, tedy třídy. Tato třída se postará o to, aby výstup byl nezávislý na způsobu komunikace a použitých nástrojů. Pokud tedy v aplikaci bude potřeba se dotazovat na server, přijatá data musí být vždy konzistentní a neměnná nezávisle na změnách, které mohou v průběhu vývoje proběhnout jak na straně serveru, tak kupříkladu při úpravách použitého protokolu a podobně. Toho lze docílit mapováním přijatých dat na další třídu, která reprezentuje přijatá data, například seznam existujících emailů.

Na obrázku 2 lze vidět jak probíhá komunikace. V tomto konkrétním případě uživatel chce zobrazit existující emaily pro určitou doménu. Controller, zde pojmenovaný *Správa emailů*, využije prostředníka, který vytvoří komunikační kanál se serverem a přijme data. Tyto data jsou přijata ve formátu specifickém pro SOAP knihovnu, jak je popsáno v kapitole 5.4. Prostředník data převede na interní formát, se kterým Controller bude umět pracovat. Data potom zobrazí uživateli ve vhodné formě.



Obrázek 2: Zobrazení existujících emailů

5.4 Knihovna kSOAP2

Aplikace využívá SOAP, který není standardně Androidem podporován. Existuje však Java knihovna pro zpracování SOAP zpráv jménem *kSOAP2*, na jejím základě byla napsána knihovna pro Android *ksoap2-android*[7] a protože je prakticky jediná, je nutno v návrhu počítat s jejími možnostmi. Tato relativně malá knihovna obstarává vlastní vytvoření SOAP dotazu, jeho odeslání na server a přijmutí odpovědi. Odpověď je naparsována do speciální třídy SoapObject, se kterou se dále pracuje. Knihovna se aktivně a rychle vyvíjí a je třeba podotknout, že v počátku psaní aplikace měla knihovna pár problémů, které v průběhu vývoje byly odstraněny. Některé z těchto problémů a jejich řešení jsou zmíněny v kapitole 6.3.

[7] `https://code.google.com/p/ksoap2-android/`

6 Implementace

V Androidu je hlavní částí každé aplikace třída Activity . Tato třída se stará o zobrazení UI a reakce na akce uživatele, např. kliknutí na tlačítko.

Poznámka 6.1 V API 11 (Android 3.0.x Honeycomb) Google přidal třídu Fragment, kterou používá třída Activity a umožňuje tak lépe aplikaci přizpůsobit různým velikostem obrazovky, nebo pro jednodušší znovupoužití kódu. Fragment je samostatný (včetně UI), ale je nutné jej použít v Activity a vyhradit pro něj místo na obrazovce pokud je to Fragment s vlastním UI, fragmentů lze najednou zobrazit i více.

Pokud chceme podporovat Fragment v nižších verzích než API 11, je třeba použít knihovnu *Support Library* ve verzi 4. Dostupná je přímo od Google v SDK Manageru.

Těchto tříd Activity je samozřejmě více a každá reprezentuje nějakou konkrétní akci, kterou uživatel zrovna provádí. Většinou co Activity to jiné „okno“ UI. Při použití třídy Fragment můžeme použít ruzné „okna“, tedy UI v jedné Activity . Například při nějaké akci, kdy je třeba provést více kroků a každý krok vyžaduje po uživateli jiné informace je elegantním řešením použít více tříd Fragment v jedné Activity . Kdy se pak Activity stará o zobrazení jednotlivých fragmentů a zpracovává data vložená uživatelem.

6.1 Android Manifest

Každá aplikace pro systém Android musí obsahovat soubor AndroidManifest.xml v kterém jsou uvedeny základní informace o aplikaci.

Mimo jiné

- určuje název balíčku ve kterém se aplikace vyskytuje, celý tento název musí být unikátní pokud chceme aplikaci distribuovat přes *Google Play*

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="cz.svethostingu.shdroid"
...
```

Výpis 6: AndroidManifest package

- popisuje komponenty aplikace – activity, services, broadcast receivers a content providers. Pokud je v aplikaci používáme, je nutné uvést cestu k třídě která je implementuje. V naší aplikaci používáme pouze Activity . U aktivit je možné nastavit různé možnosti, hlavní je ovšem možnost jejich spuštění, protože android umožňuje

spustit aktivitu externě pro nějakou konkrétní činnost. Nutné je hlavně určit jednu aktivitu jako android.intent.action.MAIN a android.intent.category.LAUNCHER, čímž ji označíme za hlavní vstupní bod a zároveň jako spouštěč.

```
...
  <activity
      android:name=".LoginActivity"
      android:label="@string/app_name"
      android:windowSoftInputMode="stateHidden"
      android:launchMode="singleTop">
      <intent-filter>
          <action android:name="android.intent.action.MAIN" />

          <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
  </activity>
...
```

Výpis 7: AndroidManifest hlavní activita

- je zde seznam oprávnění, které aplikace potřebuje pro svou funkčnost. Protože hlavní činnost aplikace je komunikace přes internet, je nutné si od systému dovolit internet používat. Další nutné oprávnění je zápis na SD kartu pro uložení stažené faktury.

```
...
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_LOGS" />
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
...
```

Výpis 8: AndroidManifest oprávnění

- informace jaká je minimální, cílená a maximální verze Androidu, které aplikace podporuje. Uvádí se číslo SDK(API). Povinně musí být uvedeno alespoň minimální podporované SDK. V příloze B jsou tyto verze vypsány.
- dále je v manifestu uvedena *ikonka* a *verze* aplikace, použité grafické téma apod.

6.2 Struktura projektu

Typický Android projekt se skládá z několika adresářů. Nejdůležitější je adresář `src/`, kde jsou uloženy zdrojové kódy, a adresář `res/`, kde jsou uloženy různé podpůrné soubory jako layouty UI, obrázky, soubory s textem pro různé jazyky a další.

Zdrojové soubory jsou v jazyce Java organizovány v balíčcích. Základní adresářovou strukturu je navržena podle použití daných tříd a jejich rozdělení do „modulů“, které oddělují třídy pro správu emailů, fakturací apod.

Základní (kořenový) balíček je `cz.svethostingu.shdroid`, ve výpisu 6 jde vidět že je uveden i v AndroidManifestu.

- `cz.svethostingu.shdroid` – v kořenovém balíčku jsou umístěny všechny používané Activity
- `cz.svethostingu.shdroid.modules` – zde se nachází další podbalíčky sdružující třídy pro konkrétní „modul“. Jsou zde fragmenty, které se zobrazují v Activitách a třídy mapující data získaná ze serveru. Třídy v balíčcích `modules` slouží převážně jako *controllery*.
- `cz.svethostingu.shdroid.soap` – balíček obsahující veškeré třídy, které pracují se sítí a SOAPem. Je zde i třída sloužící jako prostředník mezi controllery a serverovými daty popsané v kapitole 5.3.4.
- `cz.svethostingu.shdroid.tasks` – tento balíček sdružuje třídy, přes které se volají třídy z balíčku `soap` a slouží jako most mezi UI a prací se sítí. Activity nebo Fragmenty používají tyto třídy, které se postarají aby komunikace neběžela na UI vláknu a „nezasekla“ aplikaci.
- `cz.svethostingu.shdroid.global` – v tomto balíčku jsou různé třídy, které jasně nespadají do žádné kategorie. Jsou zde převážně umístěny různé utility.

6.3 SOAP

Jednou z nejdůležitějších částí aplikace je implementace komunikace pomocí SOAP protokolu. Pro tento účel je vytvořena třídu, která obsahuje statické metody jednotlivých použitelných funkcí. Těchto funkcí je značné množství a odpovídají možným příkazům posílaných serveru, např. přihlášení, získání seznamu emailů, přehledu fakturací apod. Všechny tyto metody jsou veřejné (public) a mají za úkol vytvořit instanci třídy SoapObject, do které se vloží data k odeslání na server. Tyto data jsou různá pro konkrétní funkci, ale každá funkce má své unikátní jméno, podle kterého server rozpozná dotaz.

```
public static SoapObject Login(String login, String pass) throws IOException,
    XmlPullParserException
{
    String SOAP_FUNCTION = "Login";

    SoapObject data = new SoapObject(NAMESPACE, "data");
    data.addProperty("login", login);
    data.addProperty("password", pass);

    return sendQuerry(data, SOAP_FUNCTION);
}
```

Výpis 9: Metoda pro funkci Login

Ve výpisu 9 je vidět vytvoření instance SoapObject a naplnění konkrétními daty pro fuknci *Login*. Metoda vrací výsledek privátní metody sendQuerry, výpis 10, která se stará o vytvoření HTTPS spojení, zabalení dat do SOAP protokolu, odeslání a příjem odpovědi. Tuto privátní metodu používají všechny veřejné metody. V první části metody se vytvoří instance síťového spojení HttpsTransportSE a obálky SOAP protokolu SoapSerializationEnvelope. Do obálky se vloží data s nastaveným `NAMESPACE` a názvem SOAP funkce. Dále proběhne nastavení hlavičky HTTP požadavku. Hlavička se nastavuje automaticky, ale je třeba do ní přidat nastavení kódování přes *gzip* pro zmenšení přenášených dat. To je důležité zejména kvůli *Monitoringu zátěže*, při zobrazení grafů se stahuje značný objem dat.

Komunikace probíhá metodou httpTransport.call(SOAP_FUNCTION, envelope, headers). Zde se však vyskytl problém s knihovnou *ksoap2-android*. Z přesně nezjištěných důvodů se někdy nepodařilo navázat spojení. Většinou se však podařilo na druhý nebo třetí pokus, proto je zde jednoduchý cyklus, který zkusí server zavolat třikrát. Pokud se ani potřetí nepodaří navázat spojení, vyvolaná vyjímka se propaguje výš ke zpracování. Což nakonec vyústí v informování uživatele o chybě. Tým kolem *ksoap2-android* knihovny vydával v průběhu času další verze, což subjektivně tento problém nejspíš odstranilo. Kód však byl ponechán pro případ problémů.

```
private static SoapObject sendQuerry(SoapObject data, String SOAP_FUNCTION)
    throws IOException, XmlPullParserException
{
    HttpsTransportSE httpTransport = new HttpsTransportSE(getUrl(), port, path, timeout);

    SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);

    SoapObject request = new SoapObject(NAMESPACE, SOAP_FUNCTION);
```

```
        request.addSoapObject(data);

        envelope.setOutputSoapObject(request);

        //Headers
        //gzipHeader pro zazipovani dat
        List<HeaderProperty> headers = new ArrayList<HeaderProperty>();
        HeaderProperty gzipHeader = new HeaderProperty("Accept-Encoding", "gzip");
        headers.add(gzipHeader);

        //soap se pokousi komunikovat 3x, potom vyhodi vyjimku
        int N = 3;
        for (int i = 0; i < N; i++) {
            try {
                System.setProperty("http.keepAlive", "false");
                httpTransport.call(SOAP_FUNCTION, envelope, headers);
                break;
            } catch (IOException e) {
                if (i == N - 1)
                  throw e;

            } catch (XmlPullParserException e) {
                if (i == N - 1)
                  throw e;
            }
        }
        SoapObject response = (SoapObject) envelope.getResponse();

        return response;
    }
```

Výpis 10: Metoda sendQuerry

6.4 Prostředník – MessageMediator

Jak bylo řečeno v kapitole 5.3.4, pro zprostředkování komunikace Aktivitám(controllerům) je využita jedna třída, která se postará o převedení dat na instance interních tříd. Knihovna *ksoap2-android* totiž z přijatých dat vytvoří instance třídy SoapObject a v nich další instance této třídy. Server odesílá odpověď ve formě asociativního pole, kde jednotlivé prvky jsou další pole a v nich až konkrétní data. Knihovnou je do Javy převedeno hlavní pole jako SoapObject a jeho prvky jako další instance třídy SoapObject a to jako vnitřní proměnná získatelná metodou getProperty(**int** index). Metoda *getProperty* ale vrací obecný Java Object, protože knihovna ksoap2 obsahuje i definici objektu SoapPrimitive. Ten by se měl vy-

tvořit u primitivních datových typů jako **int**, **float**, **boolean** a podobně. Jenže se tak vždy neděje a bohužel se mi nepodařilo zjistit jestli je chyba na serveru, nebo v knihovně. Ze strany serveru s tím nic moc dělat nejde, jelikož jsem byl informován že se používá standardního způsobu odeslání SOAP zprávy v PHP. Knihovna také z neznámého důvodu některé pole ze serveru převede na Vector<T>, místo na SoapObject.

S tím vším jsem musel počítat při zpracování dat ze serveru. Pokud knihovna takto vytváří a zanořuje SoapObjekty do sebe a ještě občas mění datové typy, je získání potřebných dat obtížné obyčejným cyklem, protože neznáme hloubku zanoření. Proto jsem pro tento účel vytvořil rekurzivní metodu, která v parametru dostane Object a String, tedy prohledávaná data a jméno z indexu asociativního pole. Tato metoda zjistí jaká třída je předaný Object a podle toho postupuje dále.

V případě SoapObjectu musí prohledat všechny jeho *property* a zjistit jejich typ. Pokud je typ *Map* jedná se o SoapObject, který byl vytvořen z pole. Metoda tedy zavolá rekurzivně sama sebe aby námi požadovaný String hledala v tomto objektu. Jestliže je typ *anyType* tak to znamená, že daná property je nějaký konkrétní datový typ a tedy možná i námi hledaný. Avšak je pořád instancí typu SoapObject, zavoláme metodu getProperty(String) s parametrem *"key"*. Tím získáme původní jméno v asociativním poli, které jsme získali ze serveru. Pokud hodnota „klíče" je shodná s hledaným Stringem, je tato property námi hledaná hodnota. Konkrétní hodnotu získáme metodou getProperty(String) s parametrem *"value"*, tím získáme už i přímo konkrétní datový typ, který server poslal např. String, Boolean, ...

Pokud je předaný parametr Vector<T>, který je druhem Java kolekce, je možné použít přímo metodu contains(String). Ta vrátí true pokud Vector obsahuje zadaný klíč. Podle mého názoru, by takto knihovna měla správně zpracovávat přijatá data, ale jak už jsem zmínil nevím jestli je chyba na straně knihovny nebo PHP.

6.5 Získání dat v Aktivitě

Většina implementovaných Activity, které zobrazují a reagují na UI, potřebují posílat a přijímat data ze serveru. Nemohou však využít přímo prostředníka, tedy třídu MessageMediator, protože síťová komunikace je operace blokující. Zastavila by tedy vykonávání veškerého kódu při čekání na odpověď ze serveru a aplikace se pak jeví „zaseknutá". To je samozřejmě situace nežádoucí, protože uživatel musí být schopen aplikaci ovládat za všech okolností a např. při stahování dat ze serveru aplikace zobrazí informativní dialog. Samotnou komunikaci je tedy třeba vykonat na jiném vlákně než ve vlákně v kterém pracuje

UI[8]. Starší verze Androidu, např. API 8, umožňovaly síťovou komunikaci i v UI vlákně a tedy blokování UI. V novějších verzích je toto chování přímo zakázáno a pokus o síťovou komunikaci v UI vlákně skončí vyjímkou.

Základní pravidla použití vláken, zároveň při práci s UI, jsou dvě. Jedno už bylo řečeno, *Neblokovat UI vlákno*, druhé je *Nepřistupovat k prvkům UI z jiného než UI vlákna*, tedy pracovat s komponenty UI, jako měnit text, reagovat na tlačítka apod., jen z UI vlákna, protože není takzvaně *thread-safe*. O thread-safe se lze dočíst online [8].

Možnosti v Androidu:

- třída Thread – lze použít klasickou Java třídu Thread, respektive Runnable. Vytvořit další vlákno a v něm spustit síťovou komunikaci a zpracovat odpověď ze serveru. Je však třeba dávat pozor a implementaci správně provést *thread-safe*. To zvyšuje náročnost na programátora a při komplexnějších operacích může způsobovat obtížně udržovatelný kód.

- použití AsyncTask – třída z knihoven Androidu vytvořená právě pro účel snazší komunikaci s UI vláknem. Při spuštění vykoná určený kód v jiném vlákně a o výsledku je schopná informovat v UI vlákně a tedy umožnit např. aktualizovat zobrazovaný text. Třída AsyncTask je generická, lze tedy volně nahrazovat typy vstupních a výstupních tříd použitých jako parametry nebo výsledek. Využívají se hlavně tři metody. Nejduležitější je metoda doInBackground(), která je se vykonává v jiném vláknu a vrací nějaký výsledek. Metoda onPostExecute() se spustí po dokončení doInBackground() a jako parametr přijme touto metodou vrácený výsledek. Metoda onPostExecute() už běží v UI vlákně, může tedy pracovat s komponenty UI. Třetí důležitou metodou je onPreExecute(), tato metoda je spuštěna před doInBackground () a umožní provést v UI vlákně nějakou operaci před samotným spuštěním, typicky zobrazit dialog s informací o prováděné operaci. Možnou nevýhodou třídy AsyncTask je její chování při otočení mobilního telefonu a tím způsobeným znovu spuštěním Aktivity. Tento problém je rozepsán níže.

- využití Service – komponenta aplikace podobně jako Activity , avšak bez UI a určená k běhu na pozadí i pokud aplikace neběží, tedy uživatel ji nemá v popředí. V Androidu se typicky využívá např. při poslechu hudby. Při „zavření“ hudební aplikace hudba hraje dál. Této funkčnosti lze využít i pro síťovou komunikaci. Je možné tedy implementovat vlastní Service, spustit a pak zahájit síťovou komunikaci. Tento přístup je však pro jednoduchou síťovou komunikaci našeho typu zbytečně obsáhlý. Vyžaduje

[8] http://developer.android.com/guide/components/processes-and-threads.html#Threads

běžící Service, která využívá zdroje mobilního zařízení i pokud aplikace neběží, komunikace mezi Service a Activity vyžadující data může být problémový. Standardně Service komunikuje použitím Intent, což je ve zkratce komponenta nesoucí nějakou informaci, zprávu, vyžadující akci po jiné komponentě [9]. Do Intentu lze vkládat teoreticky libovolná data, avšak třídy pro které není specificky připraven musí impementovat Serializable nebo Parcelable, více [10].

Při implementaci byl použit AsyncTask, který však nelze použít přímo v aktivitě vyžadující data. Každá aktivita má určitý „životní cyklus", lifecycle [11]. Přehled lifecycle aktivity je v příloze C obrázek 8. Tento cyklus zajišťuje správné chování aktivity, potažmo aplikace, při přerušení činnosti např. při příchozím hovoru, nebo otočení mobilu a s tím spojenou změnu rozvržení UI. Reakce na tyto vnější akce jsou různé, třeba při změně rozvržení UI je v podstatě celá aktivita zrušena a vytvořena znovu, což právě činí problematickým použití třídy AsyncTask přímo v aktivitě. AsyncTask vykonává svou činnost nezávisle na tom co se děje s aktivitou a pokud je aktivita zrušena, tedy je odstraněna z paměti, AsyncTask po dokončení činnosti spouští již zmíněnou metodu onPostExecute (), která často pracuje s nějakými prvky UI. Ty ale v té chvíli už neexistují, protože byly znovu vytvořeny a AsyncTask má tak reference na již neexistující instance a aplikace skončí s vyjímkou NullPointerException.

Možných řešení je více, např. zakázat aktivitě reagovat na otočení mobilu při použití AsyncTask. Tím nedojde k znovu vytváření aktivity a AsyncTask tedy NullPointerException nevyhodí. To ale není úplné řešení, protože znovu vytváření aktivity se může spouštět i z jiných důvodů než jen kvůli změny rozvržení UI, aplikace pak tedy zůstane náchylná k pádu. AsyncTask je tedy nějakým způsobem nutné oddělit od přímého používání UI elementů aktivity, protože ty jsou zdrojem vyjímky. Jedna z možností, a nejspíše nejjednodušší, je použití třídy AsyncTask v třídě Fragment a AsyncTask bude po, a před, dokončením činnosti volat takzvanou *callback* metodu. Třída Fragment má také určitý lifecycle, příloha C obrázek 9, avšak je podmíněný cyklu aktivity ke které je připnutý. Toto speciální použití fragmentu vyžaduje při vytváření fragmentu zavolat metodu setRetainInstance(**true**). Tím řekneme instanci FragmentManageru, aby instance na daný fragment zůstávala zachována i při znovuvytváření aktivity, lze použít pouze pokud fragment není v *back stack* [12]. Dále vytvoříme speciální *callback interface*, který musí implementovat aktivita používající daný fragment. Interface by měl obsahovat metody onPreAsyncTask() a onPostAsyncTask (), které bude volat AsyncTask používaný v fragmentu. Každý fragment totiž obsahuje metody onAttach() a onDetach(), čímž je informován o připojení, respektive odpojení od aktivity. Toto V metodě onAttach() je jako parametr předaná daná aktivita, kterou si ve fragmentu uložíme jako instanci námi definovaného *callback interface*. Třída AsyncTask

tedy po skončení a před začátkem, zavolá metodu tohoto interface, který je ve skutečnosti aktivita a ta udělá potřebné činnosti. Trik spočívá v metodě onDetach(), která se zavolá právě ve chvíli kdy je aktivity znovu vytvářena a informuje fragment o tom že byl odpojen. Proto v této metodě nastavíme uloženou instanci callback iterface na **null**. V třídě AsyncTask před použitím tohoto callbacku zjistíme jestli je **null**, pokud ne, je možné jej použít. Po znovuvytvoření aktivity FragmentManager připojuje instancované fragemnty zpátky k aktivitě, v té chvíli je zavolána metoda onAttach() kde si znovu uložíme instanci callback interface.

Protože je zavolána metoda setRetainInstance(**true**) s parametrem *true*, není fragment znovuvytvářen jako aktivita ke které je připojen a AsyncTask tedy proběhne v pořádku na pozadí. Použití této konstrukce lze vidět v příloze D, ukázkový kód je v podstatě funkční, jen někde trochu zjednodušen.

6.6 BasicFragmentActivity

BasicFragmentActivity používám jako základní třídu ze které dědí všechny ostatní aktivity. Tato třída je abstraktní, tedy nejde sama o sobě použít. Její hlavní smysl je pro sjednocení „menu“, tedy možností, které se zobrazí po stisku tlačítka na mobilním telefonu. Toto menu si každá aktivita může implementovat vlastní, ale to by znamenalo duplicitu kódu. BasicFragmentActivity tedy obsahuje kód pro základní položky menu a jejich obsluhu. Dále je zde důležitá funkcionalita ukládání globální instance aplikace. Jedná se o třídu Global, která uchovává některé informace společné pro více aktivity, např. *SSID*, *accountID*, zda se jedná o multihosting účet a podobně. Tato instance je dostupná z jakékoliv aktivity jednoduchým přetypováním metody getApplication(), protože je takto nastavena v *AndroidManifest*. Ukládáním mám na mysli zápis vnitřních proměných do *SharedPreferences* [14] a jejich načtením zpět. Ukládá se protože pokud má uživatel delší dobu aplikaci na pozadí, může se Android v rámci uvolnění „nepotřebné“ paměti rozhodnout tuto instanci smazat. Aplikace se ale nespustí znovu od aktivity přihlášení, ale tam kde uživatel naposledy skončil. Aktivita by pak chtěla číst data, která neexistují.

6.7 Implementace uživatelského rozhraní

Zde se rozepíšu o implementaci jednotlivých Activity použitých ke správě služeb v systémech svethostingu.cz. Už bylo zmíněno jak aktivity získají data ze serveru. To je ve své podstatě pro všechny aktivity skoro stejné a rutinní. Proto u jednotlivých aktivit aplikace zmíním jen odlišnosti. Pro každou serverovou funkci, nebo příkaz, je vytvořen Fragment obsahující AsyncTask. Ten potom využíváme v aktivitě. Tento fragment je však nutné

připravit buď předem, nebo spustit AsyncTask okamžitě při vytvoření fragmentu. Předem se musí připravit v případě, kdy na spuštění používáme další metodu, kterou voláme z aktivity. Připnutí fragmentu k aktivitě totiž nějakou dobu trvá a tuto činnost řídí systém. Pokud tedy hned po použití metody commit()[9] budeme chtít i spustit AsyncTask, nebude to možné. Fragment totiž ještě není inicializovaný.

V aplikaci používám oba přístupy, tedy že se AsyncTask spustí okamžitě po přidání fragmentu a nebo při zavolání metody. Záleží totiž kdy daná aktivita potřebuje tyto data. Pokud aktivita vyžaduje nějaké data okamžitě po svém spuštění, např. seznam emailů, využívám možnosti spustit AsyncTask zároveň při vytvoření fragmentu, protože je pak jednodušší jeho spouštění při vytváření aktivity. Stačí jej přidat do instance třídy FragmentManager a stahování započne automaticky. Toto stažení dat probíhá většinou jednou, proto nevadí že vyvolat znovu jde jen další instancí fragmentu.

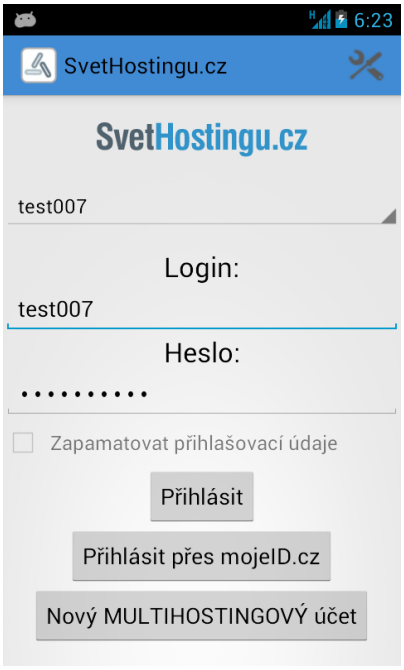
Pokud však server aplikace kontaktuje po nějaké akci uživatele, např. přidání nového emailu. Je vhodnější použít přístupu spuštění AsyncTask přes veřejnou metodu, protože pak lze odesílání dat spustit jednoduše vícekrát opětovným zavoláním metody a lze i hlídat zda odesílání dat zrovna probíhá a neduplikovat tak data, pokud uživatel třeba vyvolá odeslání nějakým způsobem vícekrát než bylo zamýšleno. Fragment pak stačí připravit, tedy přidat do aktivity při jejím vytvoření a ve chvíli kdy uživatel vyvolá akci, je tento fragment získán z FragmentManager a použit.

6.7.1 Přihlášení a výběr spravované domény

V aktivitě přihlášení, je možnost uložení uživatelského jména a hesla. Samozřejmě je potřeba při řešení tohoto problému myslet na zabezpečení, aby heslo nebylo jednoduše získatelné. Věnuji se tomu v kapitole 6.8.

Po přihlášení se spustí aktivita výběru domény, kterou bude uživatel spravovat. Výběr domény je řešen třídou Spinner, což je vlastně pojmenování klasické „roletky“. Jako zdroj dat pro Spinner je možné použít širokou škálu tříd např. standardní Java kolekce dědících z třídy List například ArrayList, nebo je možné použít teoreticky jakékoliv pole. Spinner jako datový zdroj však potřebuje třídu SpinnerAdapter. Adaptéry jsou v Androidu speciální třídy, které jsou takovým mostem mezi daty a View, v tomto případě Spinner. Při vytváření adaptéru tedy do něj vložíme data, ale nastavíme třeba i vzhled jak bude vypadat zobrazovaná položka. U výběru domény bylo nutné, aby jeden Spinner reagoval na druhý, tedy Spinner s výběrem Multihostingového účtu mění možný výběr domén v druhém spinneru, protože každý účet má samozřejmě jiné domény. Toho bylo docíleno vytvořením „posluchače“, tedy listeneru, který reagoval na změny v prvním spinneru.

[9] Metoda commit() potvrzuje transakci přidání fragmentu do aktivity



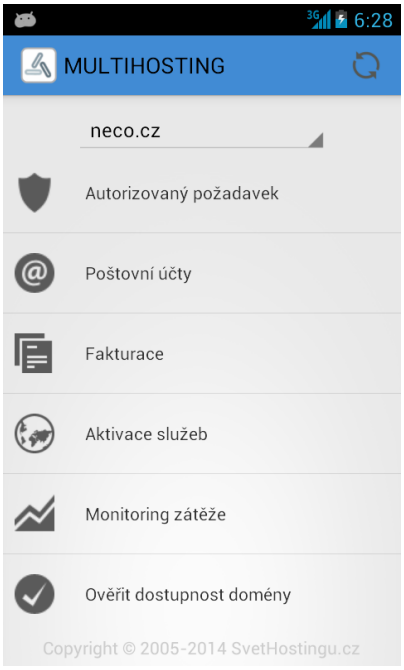
Obrázek 3: Přihlašovací obrazovka

Třída Spinner obsahuje metodu setOnItemSelectedListener(), do které lze jako parametr předat třídu, která implementuje interface AdapterView.OnItemSelectedListener. Tento interface obsahuje metody onItemSelected() a onNothingSelected, v kterých lze tedy reagovat na požadavek změny dat v druhém spinneru. Více o Spinnerech se lze dočíst na webu Androidu [13].

6.7.2 Menu administrace a autorizovaný požadavek

Zároveň se seznamem účtů pro výběr domény se stahují ze serveru i informace o uživateli. Po výběru domény se uživateli zobrazí menu se všemi možnostmi správy účtu. Jedná se o jednoduchou aktivitu a jejím účelem je spouštět jiné aktivity. V aktivitě je i možnost rychlého přepnutí mezi spravovanými doménami, bez nutnosti se vracet do aktivity předchozí. Uživatel však může během práce s aplikací přidat další domény, proto byla nutnost umožnit jejich „refresh“, tedy vyžádat a stáhnout ze serveru nový seznam domén a ten zobrazit.

Aktivita obsahující odeslání autorizovaného požadavku pouze odešle text napsaný uživatelem. Implicitně je pro odpověď nastavena emailová adresa uživatele získaná ze serveru funkcí informace o uživateli.



Obrázek 4: Menu administrace

6.7.3 Správa e-mailů

Ve správě emailových účtů jsem v rozvržení UI použil speciální view FragmentTabHost, který vytvoří jako uživatelské rozhraní „záložky“. Aktivita samotná pak spravuje tyto záložky, tedy přepíná fragmenty podle akce uživatele. Zbytek obrazovky pak zobrazuje samostatné fragmenty, které jsou nezávislé a teoreticky použitelné i mimo tuto aktivitu. Aby tato funkcionalita správně fungovala i na starších verzích Androidu bylo potřeba použít třídu z *Android SupportLibrary*. Vytvoření nového přesměrování či přeposílání je nutné vytvořit na již existující email, ale s požadavkem že na email na který existuje přesměrování logicky nemůžeme vytvořit přeposílání a naopak. Seznam existujících emailů je držen v instanci Java kolekce, konkrétně ArrayList. Lze tedy využít metodu remove(), která odstraní z ArrayListu objekt poslaný do ní jako parametr. Můžeme tedy zduplikovat kolekci a jednoduchým cyklem odstranit emaily které jsou přesměrovány či přeposílány. Takto vytvořenou kolekci dáme uživateli na výběr emailu.

6.7.4 Fakturace

Fakturace obsahuje výpis zaplacených a nezaplacených faktur. Tento seznam obsahuje několik informací, které je vhodné zobrazit hned uživateli aniž by musel otevírat de-

tail faktury. Pro tento účel jsem tedy vytvořil vlastní implementaci už výše zmíněné třídy Adapter. Tento nový adaptér jménem InvoicesSimpleAdapter, konkrétně dědí z třídy SimpleAdapter, dokáže zobrazovat čtyři různé TextView. Tedy *číslo faktury, splatnost, částka* a *info o zaplacení*. Informaci o zaplacení zobrazuje barevně, zeleně nebo červeně, podle toho zda byla faktura zaplacena nebo ne.

Detail faktury pak zobrazuje další informace, zejména přesné položky platby a dlouhý výpis údajů k platbě a fakturační adresa. Aby nemusel uživatel pořád „scrolovat" nahoru a dolů, využil jsem třídu ViewSwitcher. V layoutu je možné do tohoto view vložit další view, kdy jde vidět jen odshora první vložené. Další je možné zobrazit zavoláním metody showNext(), standardně jde tedy vidět pouze text *Zobrazit informace k platbě* a *Zobrazit fakturační adresu*. Po kliknutí na tyto nápisy je zavolána metoda showNext(), tím dojde k „prohození" view a uživatel uvidí celé texty údajů k platbě nebo fakturační adresu.

Dále je možné stáhnout PDF soubor s fakturou. To probíhá tak, že na server je posláno číslo faktury a server vrátí URL adresu odkud soubor stáhout. Samotné stažení faktury pak probíhá vytvořením instance standardní třídy URL, zavoláním metody openConnection() a otevřením streamu ke stažení openStream() do BufferedInputStream a zapsáním do souboru na SD kartu pomocí FileOutputStream. Stahování probíhá pomocí třídy AsyncTask, kde využívám její metodu publishProgress(), kterou jsem ještě nezmínil. Při zavolání této metody, kde do parametru lze vložit integer číslo, je možné během běhu AsyncTasku zavolat UI vlákno právě kvůli možnosti aktualizovat průběh činnosti, např. právě procentuální průběh stažení souboru. Po stažení určitého počtu bajtů se tedy zavolá publishProgress() a aktualizuje se stav zobrazovaného ProgressDialog na aktuální hodnotu. Po dokončení stažení je uživateli nabídnuta možnost poslat fakturu emailem nebo otevřít externí aplikaci pro PDF soubory.

6.7.5 Aktivace služeb

Implementaci více kroků k objednání služeb jak je vyžadováno v kapitole 5.1.7 je provedeno pomocí fragmentů. Každý jednotlivý krok má vlastní fragment, který má svoje UI a vlastní obslužný kód ke konkrétnímu kroku. Aktivita ve které tyto fragmenty jsou naopak drží instance proměnných. Tedy možnosti, které uživatel zvolil a údaje které zadal. Aktivita se zároveň i stará o komunikaci se serverem. V podstatě žádný fragment nepotřebuje přímo pro svou funkčnost komunikovat, protože potřebná data jako seznam TLD, uložené fakturační a doménové kontakty uživatele, se stahují při spuštění aktivity. Jednotlivé fragmenty používají ke komunikaci s aktivitou interface. Tento musí aktivita používající tyto fagmenty implementovat. Fragmenty jsou tedy použitelné v jakékoliv aktivitě, která implementuje jejich interface. Navigace mezi fragmenty probíhá tlačítky

Obrázek 5: Detail fakturace

Zpět a *Další*. Akce na těchto tlačítkách pošle přes interface zprávu aktivitě, že je potřebná změna fragmentu.

První fragment umožní uživateli vybrat typ aktivace, tedy *webhosting*, *alias s přesměrováním* nebo *alias bez přesměrování*. Výběr probíhá pomocí tří view typu RadioButton sdružených do RadioGroup. Vybraná hodnota se odešle přes interface do aktivity, kde se vyřeší uložení, tedy do nějaké proměnné. Tuto proměnnou je nutné ukládat i mezi stavy aktivity (např. při změně layoutu při otočení zařízení), k tomu se používá standardního způsobu v Androidu, tedy využití metody onSaveInstanceState(). Obdobně fungují i ostatní fragmenty zde popsané.

Další fragment je zadání domény. Uživatel tedy zadá doménu jež by si přál registrovat. K zjištění dostupnosti se pošle na server dotaz. Odpověď může mít čtyři základní možnosti a to *Domain available*, *Domain transfer*, *Domain hosted* a jako čtvrtá možnost je chyba. Domain available znamená volnou doménu, tedy nic nebrání registraci. Domain transfer znamená že je doména už zabraná, ale pokud je uživatel vlastníkem domény je možný její převod do služeb svethostingu.cz. Domain hosted není chybou, ale doména je již v systémech svethostingu.cz vedená, tedy její převod není možný. A poslední možnost, chyba, může znamenat více věcí a to například, že uživatel zadal nepodporované TLD nebo zadal doménu v nesprávném tvaru pro dané TLD. O stavu domény se uži-

vatel dozví velkým info boxem, kde je detailní popis případné chyby nebo informace. Jedná se o upravené TextView, které má nastaveno pozadí na oranžovou nebo červenou barvu podle vážnosti chyby či zelenou v případě že je doména volná. Na levé straně tohoto TextView je vykreslena ikonka v podobných barvách, ta se dá zobrazit metodou setCompoundDrawablesWithIntrinsicBounds(), která má čtyři integerové parametry každý pro jednu stranu pomyslného obdélníku zařínající zleva, tedy *left, top, right, bottom*. Barva pozadí se nastavuje metodou setBackgroundResource(). Parametrem u obou metod je ID daného resource. Pokud byl v předchozím fragmentu vybrán *alias*, je zde ještě Spinner. Ten má jako data nastaveny všechny existující domény v aktuálně přihlášeném účtu. Ty slouží pro výběr *hlavní domény*, tedy na kterou právě vytvářená doména bude odkazovat.

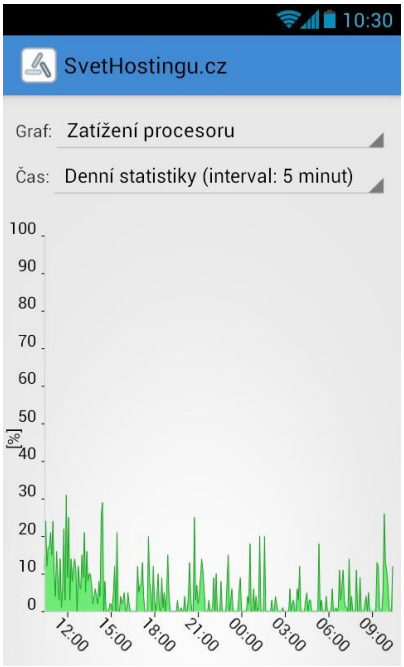
Následující fragment se odvíjí podle toho zda je zvolená doména volná k registraci nebo zda je vhodná pro transfer jak bylo popsáno v kapitole 5.1.7. Pro transfér je možnost aktivace a převedení, nebo jen aktivace. Volba probíhá pomocí RadioButton.

Pokud je doména volná k registraci následuje fragment s volbou doménového kontaktu. Uživateli se zobrazí klasický formulář požadující jméno, příjmení, adresu atd. Při spuštění aktivity se stáhly uložené kontakty, které jsou uživateli nabídnuty ve Spinner view a po jejich vybrání se formulář předvyplní. Pokud uživatel žádné uložené kontakty nemá nebo pokud chce vytvořit nový konktakt, formulář je dostupný pro editaci. Před možností pokračovat dál probíhá validace vložených údajů. Testuje se správný tvar emailu, tel. čísla a zadání základních informací. Email a tel. číslo se kontrolují pomocí regulárních výrazů, pro které má Java, respektive Android, zabudované funkce. Každý String má metodu matches() do které jako parametr lze předat regulární výraz, kterému musí string odpovídat.

Poté následuje fragment se zadáním fakturačním kontaktem. Ten funguje obdobně jako fragment s doménovým kontaktem, rozdíl je v tom že se zobrazuje vždy. Tedy i při převodu domény, kdy se doménový kontakt kopíruje z existující domény.

Jako poslední je uživateli zobrazen fragment rekapitulace. Zde jsou uvedeny všechny volby, které uživatel zvolil a případná cena, tedy platba, za prováděné operace. Uživatel má možnost objednávku potvrdit a odeslat nebo přidat do košíku a spustit další objednávku, případně objednávku zrušit.

Speciálním případem je košík objednávek. Seznam všech objednávek je uložen v nějaké proměnné v aktivitě. V kterékoli části aktivity lze tento košík vyvolat tlačítkem v menu. Nové verze Androidu zobrazí toto tlačítko v takzvaném *ActionBaru* jako ikonku košíku. Košík je řešen pomocí třídy Dialog, která má vlastní layout. Layout se skládá z tlačítka pro odeslání objednávky a pro vymazání košíku a hlavně tedy zobrazení všech objednávek v košíku pomocí ListView. Pro toto ListView jsem vytvořil třídu BasketAdapter pro snazší



Obrázek 6: Graf zatížení procesoru v intervalu pět minut

zobrazení položek košíku. Dialog je inicializován se speciální *téma* v kterém je nastavena animace zobrazení. Samotný dialog má gravitaci na spodní stranu obrazovky a zobrazuje „vysunutním“ zespodu nahoru. Při zavření se zase dolů „zasune“.

Po odeslání objednávky je uživatel informován dialogovým oknem o úspěchu aktivace služeb.

6.7.6 Monitoring zátěže

V této aktivitě jsou uživateli zobrazeny v grafické formě různé statistiky týkající se jeho multihostingu, které jsou popsány v kapitole 2. Aktivita při svém spuštění stáhne ze serveru všechna potřebná data. Data jsou zde *timestamp*, tedy čas, a *value*, tedy hodnota představující např. zatížení procesoru a podobně podle typu grafu.

Pro vykreslení grafů jsem zvolil knihovnu *AChartEngine*[10]. Tato volně dostupná knihovna ma různé možnosti zobrazení grafů, spojnicový, sloupcový, koláčový atd. V aplikaci je potřeba zobrazit spojnicový a koláčový. Koláčový graf je využit u zobrazení obsazenosti disku. Graf se v aktivitě vykresluje pomocí použití fragmentu. Fragment má svůj layout do kterého zobrazí graf a aktivita ve svém layoutu nad grafem zobrazí dva Spinnery. Jeden pro výběr grafu a druhý pro časový interval. Jejich změnou se tedy mění zobrazovaný

[10] `http://achartengine.org/`

graf. Samotná knihovna je velmi dobře zpracovaná a v základu funguje vytvořením třídy TimeSeries, tedy nějaká data v čase, které se vloží do třídy XYMultipleSeriesDataset tyto data sdružující. Pokud bychom v jednom grafu zobrazovali více křivek, postupně všechny jejich TimeSeries vložíme do XYMultipleSeriesDataset. V aplikaci je ale v jednu chvíli vždy zobrazena jen jedna křivka. Tento *dataset* je potom vložen do takzvaného *renderu*, instanci třídy XYMultipleSeriesRenderer. Časový údaj se vkládá ve formátu Timestamp a knihovna jej pak umí překonvertovat do libovolného formátu, který je čitelný pro člověka. Knihovna má opravdu spoust dalších nastavení, které by však vyžadovaly více prostoru než je v této práci možno věnovat.

6.7.7 Ověření dostupnosti domény

Tato aktivita je primárně určena pro rychlé vyhledání dostupnosti domény. Funguje v podstatě stejně jako fragment v části Aktivace služeb pro vložení domény s tím rozdílem, že jen uživatele informuje zda je doména volná či nikoliv. V případě že je volná navíc nabídne možnost její registrace. Pokud uživatel tuto možnost použije otevře se aktivita spravující aktivaci služeb.

6.8 Zabezpečení ukládání přihlašovacích údajů

Tato kapitola se věnuje zabezpeční v Androidu ve smyslu uložení jména a hesla uživatele při přihlašování. Z logických důvodu nelze jméno a hlavně tedy heslo ukládat přímo jako text někam do souboru. Jaké jsou tedy v Androidu možnosti. Nejdřív je třeba podotknout, že ukládání jména a hesla jsem v aplikaci řešil jako jednu z prvních věcí na začátku vývoje, tedy někdy v první polovině roku 2012. Nemusí zde být tedy zmíněny všechny možnosti které jsou v Androidu dostupné dnes.

6.8.1 Možnosti ukládání dat v Androidu

- Shared Preferences – základní forma ukládání dat ve formě párů klíč-hodnota. Reálně Android vytvoří XML soubor v části paměti telefonu přístupné pouze aplikaci která jej vytvořila.
- Internal Storage – Android vývojáři zpřístupní přímo paměti v telefonu místo kam může vytvořit libovolný soubour
- SQLite Databáze – uložení dat do SQLite souboru. Databáze je přístupná jen aplikaci která ji vytvořila.

- External Storage – podobně jako Internal Storage, pouze se data ukládají na „externí“[11] SD kartu.

Z výše uvedených možností se jako naprosto nevhodné jeví External Storage a to z důvodu ukládání na externí a tedy uživateli přístupné médium. Co se týká zbylých možností, z pohledu bezpečnosti jsou na tom stejně, protože data které vytvoří existují někde uvnitř systému a tedy nejsou běžně dostupné uživateli. Pokud uživatel na telefonu neprovedl takzvaný „root“, tedy získání administrátorských práv v celém systému Android a tím možnost do těchto adresářů nahlížet a tyto soubory číst.

Pro ztížení situace případnému zloději uživatelských účtů jsou ukládaná jména a hesla šifrována. Šifrování probíhá pomocí knihoven v balíku `javax.crypto`, konkrétně třídy `Cipher` a algoritmu *AES*. Přihlašovací údaje se šifrují pomocí hesla, které je uloženo v kódu šifrovací třídy. Případný útočník by musel tedy dekompilovat celou aplikaci, najít tuto třídu a použít heslo na ukradené údaje z interní paměti telefonu, což považuji za dostatečné zabezpečení. Pro vyšší zabezpečení je možnost generovat toto heslo na základě nějakého unikátního řetězce pro každý telefon. Ovšem dekompilací se tento algoritmus dá zjistit také, takže by tato technika způsobila jen různá hesla, ale pokud by o ní útočník věděl jistě si dokáže obstarat tyto unikátní údaje pokud dokázal vytáhnout soubor z paměti telefonu určené jen pro jednu aplikaci.

Vlastní ukládání tedy provádím do *Shared Preferences*. Jména a hesla se spojí do jednoho dlouhého Stringu ve formánu „login—username“ a takto opakovatelné za sebou více údajů každé slovo odděleno znakem „—“. Protože uživatelé mohou mít tento znak ve svém heslu, je potřeba prvně všechny tyto znaky nahradit zástupným znakem a po přečtení zase nazpátek vrátit původní znak. Tento dlouhý String je zašifrován a uložen.

6.9 Implementace mojeid.cz

Služba mojeid.cz je poskytována organizací CZ.NIC a v zásadě umožňuje mít jeden uživatelský účet pro více webů. Svethostingu.cz tuto službu integruje, ale služba samotná přímo Android nepodporuje. Abychom umožnili uživateli se přihlásit i touto cestou, využili jsme možnosti chytrých telefonů a to v dnešní době celkem bezproblémového zobrazení webu. Android umožňuje vytvořit přímo v aktivitě webový prohlížeč aniž by uživatel musel otevírat externí. Tím má vývojář kontrolu nad daty, které uživatel vidí, na jaké je webové adrese apod. Na autentizačním serveru služeb svethostingu.cz byla vytvořena webová stránka, která požaduje v URL jako parametr uživatelské jméno, to

[11] Úložiště nemusí být skutečně externí, ve většině zařízení se jako external storage počítá i interní paměť telefonu. Tato paměť však není Internal Storage.

následně přepošle standardní cestou službě mojeid.cz. Ta zobrazí stránku pro přihlášení uživatele a uživatel se tedy normálně přihlásí do služby mojeid.cz tak jakoby se přihlašoval z kteréhokoli jiného webu, který tuto službu podporuje.

Po přihlášení si autentizační server svethostingu.cz ověří identitu a pokud je vše v pořádku přesměruje prohlížeč na „fiktivní" URL. Fiktivní pouze zdánlivě, protože je v ní obsažená určitá informace v parametrech, které nejsou pro server důležité. V aktivitě totiž lze kontrolovat na jaké adrese se právě prohlížeč nachází, nabízí se tedy kontrolovat určitou přesně danou adresu a pokud se vyskytne, aktivita pozná že byl uživatel přihlášen nebo přihlášení selhalo.

Tato url vypadá nějak takto soap.svethostingu.cz/openid_shdroid.php?li=abcdf. V parametru *li* je uloženo SSID, který se použije pro další komunikaci a autentizaci toho že je uživatel přihlášen. Hodnotu parametru lze získat metodou getQueryParameter("li"), která vrací String.

Takto lze implementovat službu mojeid.cz v Androidu pokud už je implementována na serveru a server tedy slouží jako prostředník pro komunikaci a autentizaci.

7 Testování

Pro správné testování na Android platformě je nutné testovat aplikace na co nejširším možném okruhu různých zařízení. Pro testování této aplikace je nejdůležitější otestovat vzhled na různých velikostech displejů a chování v různých verzích Androidu. Společnost Google k tomuto účelu poskytuje emulátor, který je na vysoké úrovni a opravdu se chová jako reálné zařízení. Ovšem testování na reálném zařízení je nutné provést také. Aplikace byla testována na třech různých reálných zařízeních. Vývoj a hlavní testování probíhalo na *Huawei Ascend Y300* s čtyř palcovým displejem s rozlišením 480x800. Uživatelské rozhraní aplikace je však navrženo pro všechny velikosti a rozlišení.

Během testování jsem zjistil, že síťová komunikace probíhá relativně rychle a to i při mobilním datovém připojení.

Při testování se aplikace připojovala na testovací server, kde nebyly použity reálná data. S použitím tohoto serveru jsem našel nejvíce chyb v kódu aplikace. Po tomto odladění jsme přistoupili k širšímu testování na reálných datech s reálným serverem a aplikaci zpřístupnili do uzavřeného testování pro pár zařízení.

8 Závěr

S prací na této aplikaci a na platformě Android jsem začal s malými znalostmi o této problematice. Během tvorby jsem do tématu pronikl více než dostatečně a byl jsem schopen překonat problémy, které vznikly s knihovnou *ksoap2-android* a další problémy popisované v této práci. Také jsem pronikl do světa hostování webů a způsobu jakým tyto služby poskytuje firma SvetHostingu.cz.

Architektura aplikace se drží vzoru MVC a odděluje komunikaci a zpracování dat od jejich prezentace. V části aplikace komunikující se serverem jsem musel překonat problémy při zpracování dat a zajistit aby výstup z této části byl konstantní a stále stejně použitelný při zobrazení dat uživateli. To zajišťuje třída prostředníka, kterou používají všechny části aplikace vyžadující komunikaci se serverem. Pro správné fungování síťové komunikace jsem vytvořil konstrukci umožňující zachování probíhající komunikace i při otočení displeje a tím způsobené znovuvytváření Acticity.

Vývoj stále probíhá, protože doposud nebyly implementovány všechny funkce, které jsou dostupné v administraci SvetHostingu.cz. V přípravě je administrace databází pro které už nějaký kód v aplikaci existuje, ale není přístupný. Do budoucna se plánuje přidat správa CRONu a úprava DNS záznamů.

Aplikace je dostupná ke stažení na Google Play a od uživatelů obdržela kladná hodnocení a recenze.

9 Reference

[1] *JSON* [online]. [cit. 2014-04-17]. Dostupné z: http://json.org/

[2] Extensible Markup Language (XML) 1.0 (Fifth Edition). *World Wide Web Consortium (W3C)* [online]. 2008 [cit. 2014-04-17]. Dostupné z: http://www.w3.org/TR/xml/

[3] Latest SOAP versions. *World Wide Web Consortium (W3C)* [online]. 2007 [cit. 2014-04-17]. Dostupné z: http://www.w3.org/TR/soap/

[4] MURPHY, Mark L. *Android 2: průvodce programováním mobilních aplikací*. Vyd. 1. Překlad Jakub Mužík. Brno: Computer Press, 2011, 375 s. ISBN 978-80-251-3194-7.

[5] ALLEN, Grant. *Android 4: průvodce programováním mobilních aplikací*. 1. vyd. Překlad Jakub Mužík. Brno: Computer Press, 2013, 656 s. ISBN 978-80-251-3782-6.

[6] *Android Developers* [online]. 2009 [cit. 2014-04-20]. Dostupné z: http://developer.android.com/

[7] *Apple Developer* [online]. 1999 [cit. 2014-04-20]. Dostupné z: https://developer.apple.com/

[8] A Short Guide to Mastering Thread-Safety. *Thinking Parallel* [online]. 2006 [cit. 2014-04-20]. Dostupné z: http://www.thinkingparallel.com/2006/10/15/a-short-guide-to-mastering-thread-safety/

[9] Intents and Intent Filters. *Android Developers* [online]. 2009 [cit. 2014-04-20]. Dostupné z: http://developer.android.com/guide/components/intents-filters.html

[10] Parcelable vs Serializable. *Developer Phil* [online]. 2013 [cit. 2014-04-20]. Dostupné z: http://www.developerphil.com/parcelable-vs-serializable/

[11] Activities. *Android Developers* [online]. [cit. 2014-04-20]. Dostupné z: http://developer.android.com/guide/components/activities.html#Lifecycle

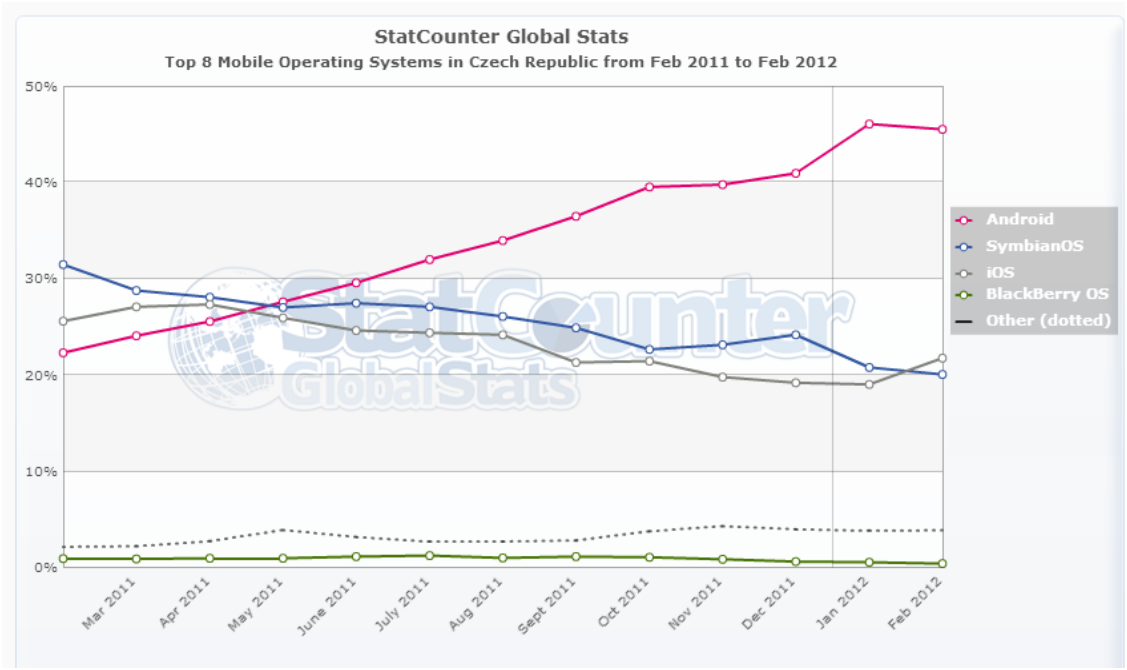
[12] Tasks and Back Stack. *Android Developers* [online]. [cit. 2014-04-21]. Dostupné z: http://developer.android.com/guide/components/tasks-and-back-stack.html

[13] Spinners. *Android Developers* [online]. [cit. 2014-04-21]. Dostupné z: http://developer.android.com/guide/topics/ui/controls/spinner.html

[14] Storage Options. *Android Developers* [online]. [cit. 2014-04-21]. Dostupné z: http://developer.android.com/guide/topics/data/data-storage.html#pref

A Podíl na trhu mobilních operačních systémů v ČR v roce 2012

Obrázek 7 ilustruje podíl na českém trhu v intervalu jednoho roku podle serveru statcounter.com [12]. Graf ukazuje silný nárůst podílu ve prospěch systému Android.



Obrázek 7: Podíl na trhu mobilních operačních systémů v ČR 2011–2012

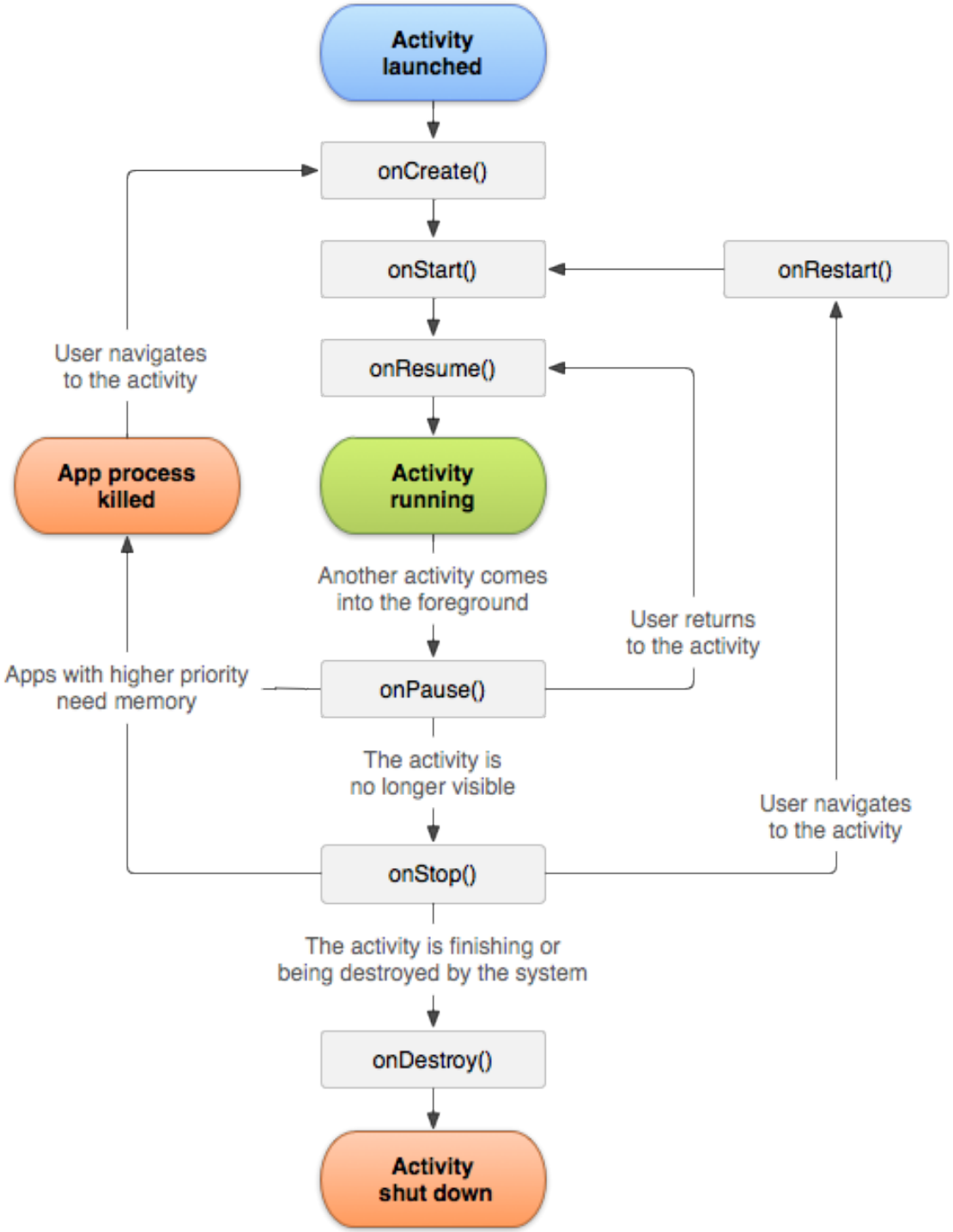
[12] http://gs.statcounter.com/#mobile_os-CZ-monthly-201102-201202

B Přehled verzí Androidu

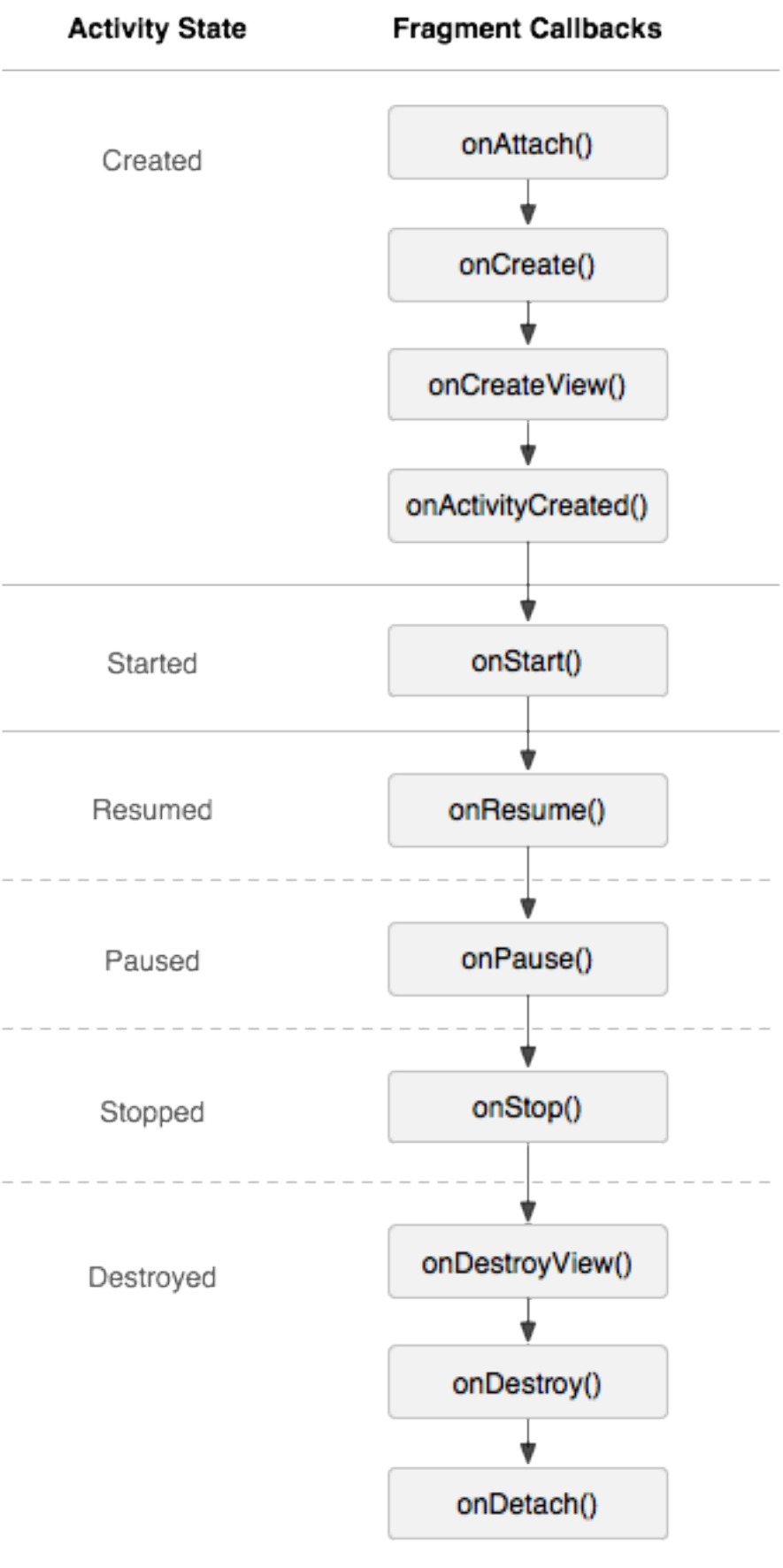
Zde je přehled hlavních verzí Androidu a jejich využití k 3. 3. 2014. Verze s distribucí menší než 0,1% nejsou zahrnuty.

Verze	Codename	API	Distribuce
2.2	Froyo	8	1,2%
2.3.3 – 2.3.7	Gingerbread	10	19,0%
3.2	Honeycomb	13	0,1%
4.0.3 – 4.0.4	Ice Cream Sandwich	15	15,2%
4.1.x	Jelly Bean	16	35,3%
4.2.x		17	17,1%
4.3		18	9,6%
4.4	KitKat	19	2,5%

C Lifecycle Activity a Fragmentu



Obrázek 8: Životní cyklus Activity



Obrázek 9: Vliv Activity na Fragment

D Ukázka použití třídy AsyncTask v Activity

```
public class LoginTaskFragment extends Fragment{
  public interface LoginCallback{
    void onPreLoginExecute();
    void onPostLoginExecute(Status result);
  }

  private LoginCallback callback;
  @Override
  public void onAttach(Activity activity ) {
    super.onAttach(activity);
    callback = (LoginCallback) activity;
  }

  @Override
  public void onDetach() {
    super.onDetach();
    callback = null;
  }

  @Override
  public void onActivityCreated(Bundle savedInstanceState) {
    super.onActivityCreated(savedInstanceState);
    setRetainInstance(true);
  }

  public startTask(String login, String pass){
    LoginAsyncTask task = new LoginAsyncTask();
    task.execute(login, pass);
  }

  private class LoginAsyncTask extends AsyncTask<String, Void, Status> {

      @Override
      protected void onPreExecute() {
        if (callback!=null)
            callback.onPreLoginExecute();
      }

      @Override
      protected Status doInBackground(String... params) {
        return new MessageMediator().login(params[0], params[1]);
      }

      @Override
      protected void onPostExecute(Status result) {
        if (callback!=null)
            callback.onPostLoginExecute(result);
      }
  }
}
```

Výpis 11: Fragment s třídou AsyncTask

```
public class LoginActivity  implements LoginTaskFragment.LoginCallback{
    ...
    private void prepareLoginFragment(){
        LoginTaskFragment fragment = new LoginTaskFragment();
        FragmentManager fm = getFragmentManager();
        fm.beginTransaction().add(fragment, FRAGMENT_TAG).commit();
    }
    private void login(String login, String pass){
        LoginTaskFragment fragment = (LoginTaskFragment)getFragmentManager().
              findFragmentByTag(FRAGMENT_TAG);
        fragment.startTask(login, pass);
    }
    ProgressDialog dialog;
    @Override
    public void onPreLoginExecute(){
        dialog = new ProgressDialog();
        dialog.setIndeterminate(true);
     dialog.setMessage(getString(R.string.login_message));
     dialog.show();
    }
    @Override
    public void onPostLoginExecute(Status result){
        if(dialog != null)
            dialog.dismiss();

        if(result.isStatusOK())
            nextActivity();
        else
            showLoginFailed(result);
    }
    ...
}
```

Výpis 12: Activity používající AsyncTask